# Minerva: Learning to Infer Network Path Properties

Rita H. Wouhaybi*, Puneet Sharma†, Sujata Banerjee† and Andrew T. Campbell‡

* Department of Electrical Engineering, Columbia University, New York NY
(now at Intel Research, Intel Corporation, Hillsboro OR)
Email: rita.h.wouhaybi@intel.com
†HP Labs, Palo Alto CA
Email: {puneet.sharma, sujata.banerjee}@hp.com
‡Dartmouth College, Hanover NH
Email: campbell@cs.dartmouth.edu

***Abstract*—Knowledge of the network path properties such as latency, hop count, loss and bandwidth is key to the performance of overlay networks, grids and p2p applications. Network operators also use these metrics for managing and diagnosing problems in their networks. However, the size of the Internet makes the task of measuring these metrics immensely difficult. A more scalable approach of inference and estimation of these metrics based on partial measurements has been recently adopted.**

**Current inference approaches do not adapt to different network topologies and the evolution of the network over time. In this paper, we propose a novel *learning* based approach, called *Minerva*, for the inferencing of inter-node properties. Minerva uses partial measurements to create signature-like profiles for the participating nodes. These signatures are later used as input to a trained Bayesian network module to estimate the different network properties. We have built a system based on our approach and present performance results from real network measurements obtained from the Planet-Lab testbed. The sensitivity of the system to different parameters including training set, measurement overhead, and size of network have also been studied in this paper.**

## I. Introduction

A number of emerging popular applications are being deployed using overlay networks which require the knowledge of end-to-end network metrics such as latency, hop count, bandwidth and loss. Driven by the measurement overheads for the scale of the Internet, the research community has produced numerous systems that attempt to estimate and infer network metrics and vital signs using partial measurements. Given the deterministic nature of the heuristics used in most of the existing inference mechanisms, they fail to perform accurately over a diverse set of network topoplgies. Similarly, these approaches can not incorporate the time based variation of metrics or the evolution of network topologies over time. In this paper, we make a case for the use of machine learning techniques and probabilistic methods in the inference of network path properties. We have developed a new inferencing framework and built a prototype, called *Minerva*, that can learn the latent characteristics of diverse network topologies not captured by deterministic methods.

It must be noted that we are not proposing yet another new inference algorithm; instead we propose a new methodology that can also improve existing algorithms to work well on a range of network topologies. Our method consists of extracting signature-like profiles for nodes from a limited number of measurements. As we explain later, the signature generation can be based on the use of similar heuristics as used by the existing inferencing algorithms. A subset of signatures and truth values is used to train a learning-based system, based on Bayesian networks [6], [8], which can then be used to infer all-pair properties. We show through experimental results that such a learning-based approach provides superior results when compared to existing systems. Our methodology also adapts to changes in the underlying topology and is able to incorporate time of day as an input in the inference mechanism. Though the focus of this paper is on inference for network latency and proximity between nodes, our methodology can be easily applied to other metrics of interest such as loss, bandwidth, etc. The paper proceeds with the discussion of related work in Section II. We then describe our methodology and system in Section III. In Section IV, we discuss the summary of the evaluation data and the results obtained. We finally conclude in Section V.

## II. Related Work

In recent past, two scalable services, *iPlane* [9] and $S^3$ [23], have been deployed for providing Internet path properties. Such services rely on tools and algorithms for scalably inferencing Internet path properties. In this section, we review only the techniques to estimate network distances and proximity since we apply the learning based approach to estimate these metrics.

NetQuest [18] proposes a framework for selecting the type of measurements needed in order to provide maximum information needed to infer properties. Even though both Minerva and NetQuest use Bayesian learning to achieve better inference of network metrics, our approach is different as it extends over existing systems, as we show in the following sections, instead of creating a whole new framework. The reasoning behind our solution is mainly due to the fact that we wanted to extend on systems that researchers and network administrators might be comfortable dealing with. Doing so we show the advantage of using machine learning in general, and Bayesian learning more specifically, to improve the performance of existing familiar systems instead of revamping the whole infrastructure.

There are schemes that use landmark techniques for network distance estimation. Landmark schemes such as Global Network Positioning (GNP) [11] use a node's distances to a common set of landmark nodes to estimate the node's physical position. In these schemes the nodes conduct measurements to every landmark node. The intuition behind such techniques is that if two nodes have similar latencies to the landmark nodes, they are likely to be close to each other. In GNP, landmark nodes measure the Round Trip Times (RTTs) among themselves and use this information to compute the coordinates in a Cartesian space for each landmark node. These coordinates are then distributed to the clients. The client nodes measure RTTs to the landmark nodes and compute the coordinates for themselves, based on the RTT measurements and the coordinates of the landmark nodes it receives. The Euclidean distance between nodes in the Cartesian space is directly used as an estimation of the network distance. GNP requires that all client nodes contact the same set of landmarks nodes, and the scheme may fail when some landmark nodes are not available at a given instant of time. The Lighthouse [13] scheme addresses this problem.

Despite the variations, current landmark techniques share one major problem. They cause *false clustering* where nodes that have similar landmark vectors but are far away in network distance are clustered near each other.

Vivaldi [3] is another scheme that assigns a coordinate space for each host, but it does not require any landmarks. Instead of using probing packets to measure latencies, it relies on piggybacking probes on data packets when two hosts communicate with each other. With the information obtained from passively monitoring packets (e.g., RPC packets), each node adjusts its coordinates to minimize the difference between estimates and actual delay. Although Vivaldi is fully distributed, it takes time to converge, requires applications to sample all nodes at relatively same rate to ensure accuracy, and expects packets to add Vivaldi-specific fields.

*Netvigator* [17] is an attempt to leverage triangular inequality and improve the performance of landmark-based measurements. Instead of ping measurements, each node conducts traceroutes to selected landmark nodes. Thus each node collects the distance information not only to the andmarks but also to the intermediate routers. The nodes (including the end-nodes, landmarks and intermediate routers) that appear in multiple traceroutes are called *milestones*. A triangular inequality based clustering heuristic, called *Min-Sum*, utilizing the distance between the nodes and milestones is used to estimate the distance between various nodes and the milestones. Hence, *Min-Sum* is an upper bound on the distance between the various nodes. While the performance results from PlanetLab measurements are promising, the tightness of this upper bound is dependent on the coverage of the underlying topology by the traceroute measurements. In this paper we use *Min-Sum* as a candidate for comparing the performance of our approach.

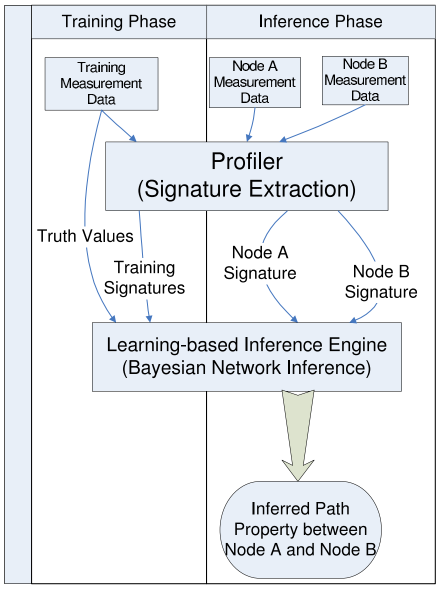
Training Phase
Inference Phase
Training Measurement Data
Node A Measurement Data
Node B Measurement Data
Profiler (Signature Extraction)
Truth Values
Training Signatures
Node A Signature
Node B Signature
Learning-based Inference Engine (Bayesian Network Inference)
Inferred Path Property between Node A and Node B

Fig. 1. Minerva Architecture

## III. Learning-based Inference Methodology

All of the inferencing techniques described in the previous section lack applicability to diverse network topologies as well as adapatability to time variation of the network metrics. This requires nodes to repeat their measurements continuously, thereby increasing the measurement overhead to ensure more accurate results. We now describe *Minerva*, a new approach that leverages learning based techniques to increase adapatability of the network property inferencing algorithms. *Minerva's* inferencing engine can be trained to achieve a degree of "expertise" and adapt to the changes in network topology as well as time-based variations of network metrics. We believe that such a system can greatly improve the accuracy of existing inferencing algorithms. Figure 1 shows the two main components of our system, namely, the **Profiler** and the **Learning-based Inference Engine**. Figure 1 also shows the flow-chart for the inference mechanism. Like existing inferencing tools, our goal is to infer the path properties of the complete network using only selected partial measurements. As we mentioned earlier, *Minerva* uses measurements similar to those of the deterministic algorithms. In the system presented in this paper each node conducts traceroute measurements only to a set of selected landmarks, like the landmark-based approaches, such as *NetVigator* [17]. The function of the profiler is to take these node measurements and extract signature-like profiles for each node. These signatures attempt to capture the topological relationships among different nodes in the network. Just like the heuristics used in deterministic inference algorithms, the signature generation schemes are based on the known invariants in the network.

As shown in Figure 1, the inferencing engine takes the node signatures generated by the profiler as input to estimate the path properties between any two nodes. Intially, the inferencing module undergoes a training phase. The training measurement data consists of a subset of the node measurements for signature generation as well as true end-to-end measured values of the metrics of interest. Thus the measurement overhead of *Minerva* is marginally higher than the overhead of similar deterministic schemes. During the training phase, both the signatures and true values are provided to the learning engine. Based on the training, the inference engine can learn about the *latent* dependencies in the system. Once trained, the inference engine can take the signatures from any two nodes, say node $A$ and node $B$, to provide a final estimate for the network path property between the nodes $A$ and $B$. The inference engine might need to be retrained as the size of the network grows significantly. It is the training ability of Minerva, that increases the adaptability to diverse network topologies. We believe *Minerva* can be used for estimating various path properties. In this paper, we evaluate its performance for inferring latency (as a QoS parameter) and the number of hops (as a topological view), among the nodes. Studying and evaluating other metrics is part of ongoing research.

As described in the Section II, many systems target latency estimation. Given the higher accuracy of Netvigator, we picked it for evaluating the performance of Minerva in estimating path latencies. Netivgator uses *Min-Sum* as its basic algorithm for estimating inter-node latency. We are not aware of any mechanism or algorithm proposed for hop count estimation. Thus, we modify the Min-Sum algorithm, used for latency estimation in [17], in order to estimate hop count in addition to latency and use it for comparison purposes. When we evaluate our algorithm for latency estimation, we also compare it to Vivaldi [3]. Before describing our system, we start with a brief description of the Min-Sum [17] algorithm so as to introduce various terms. We then describe our profiling algorithm followed by Bayesian techniques used in our inference engine.

## A. Min-Sum Algorithm

As mentioned earlier, the Min-Sum algorithm proposes estimating network latencies among nodes using heuristics based on triangular inequality. We now provide a short summary of its operation.

In a system with $N$ nodes and $L$ landmarks, each node conducts traceroute measurements to every landmark. We refer to these measurements as the discovery of the uplink routes. In addition, if we are considering the asymmetric Min-Sum algorithm where routes on the network can be asymmetric, then each landmark will also conduct traceroute measurements to every node on the network. We refer to this set of measurements as the discovery of the downlink routes. The result is $2 * N * L$ measurements. Every time a router is encountered more than once, in the traceroute measurements, its status is “promoted” to milestone. Note that the definition of a router includes the landmarks themselves, even if they are, physically, servers or end-nodes. We denote the set of common milestones encountered on the uplink routes from node $i$ and the downlink routes to node $j$ as $L(i, j)$. The Min-Sum algorithm then estimates the distance between a node $i$ and a node $j$ as:

$$min(dist(i, l) + dist(l, j)), \forall l \in L(i, j), \qquad (1)$$

where $dist(i, l)$ is the distance from node $i$ to milestone $l$, whether that distance is latency or hop count. In fact, considering the intuition of triangular inequality, the Min-Sum algorithm provides an upper-bound estimate for network latency among nodes.

```
**Calculate Node Histogram Profile** { // from $i$ to $j$
    obtain $\mathcal{M}_{i,up}$ & $\mathcal{M}_{j,down}$;
    calculate distances $\mathcal{D}_{i,up}$ from $i$ to $\mathcal{M}_{i,up}$;
    calculate distances $\mathcal{D}_{j,down}$ from $\mathcal{M}_{j,down}$ to $j$;
    map $\mathcal{D}_{i,up}$ to a histogram $\mathcal{H}_{i,up}$;
    map $\mathcal{D}_{j,down}$ to a histogram $\mathcal{H}_{j,down}$;
    $P_{i,j} = [\mathcal{H}_{i,up}, \mathcal{H}_{j,down}]$;}
```

Fig. 2. Node Histogram Profiling Algorithm Pseudocode

## B. Profiling

As mentioned earlier, the profiler extracts node signatures from the measurements conducted by each node. The signatures, by design, do not carry the explicit identity of the node in question. By doing so, we aim at creating an inference engine that adapts with the dynamics of the network related to nodes joining and leaving, where signatures can sufficiently reflect nodes behaviour without attaching an identity of a specific node to a profile. This idea draws similarity from the approach used in detecting worms on the Internet by creating signatures of their behaviour. Though we explored several signature-generating algorithms, we only discuss and present results from one scheme called *Node Histogram*. We now describe the operation of the *Node Histogram* algorithm, a summary of its pseudocode is presented in Figure 2. The Node Histogram profiling algorithm is designed to retain topological information about the position of nodes with respect to all milestones encountered with traceroute measurements. When conducting measurements to landmarks, a node $i$ encounters a set of milestones that we denote by $\mathcal{M}_{i,up}$. The distances to these milestones is represented by the vector $\mathcal{D}_{i,up}$. Node $i$ converts $\mathcal{D}_{i,up}$ into a histogram that we denote by $\mathcal{H}_{i,up}$. Lets consider an example network with a diameter of 12 hops. Assume it has a Node $x$ with the following distances to milestones vector $\mathcal{D}_{i,up} = [2, 2, 3, 5, 6, 6, 6, 8, 10]$. Mapping this vector into a 12-dimensional histogram, we obtain $\mathcal{H}_{i,up} = [0, 2, 1, 0, 1, 3, 0, 1, 0, 1, 0, 0]$. Note that the histogram starts with 1 as the minimum distance. In the above example, since we had no milestone that is 1 hop away from Node $x$, we set the first value to 0. However, we have two milestones that are each 2 hops away, thus, we set the second value to 2, and so on. In our implementation, $\mathcal{H}_{i,up}$ is a 32-dimensional vector, representing the maximum number of hops as defined in traceroute measurements. Similarly, a histogram is built for the downlink measurements for every node denoting the distances from the milestones to the node. We denote the downlink

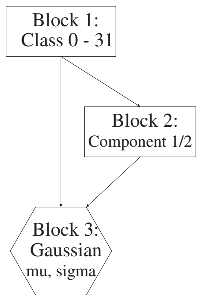

Fig. 3. Simple Bayesian Network Structure

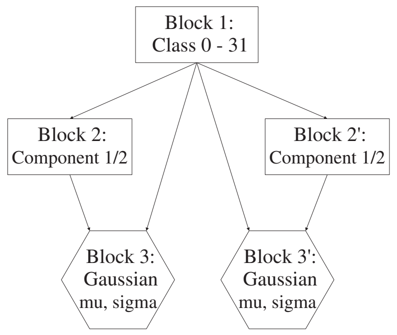

Fig. 4. Modified Bayesian Network Structure

histogram vector by $\mathcal{H}_{i,down}$. Thus, the input to the Bayesian module consists of the concatenation $[\mathcal{H}_{i,up}, \mathcal{H}_{j,down}]$ when estimating the distance from node $i$ to node $j$.

Visualizing the Node Histogram profiling algorithm, if a node sits in the center, the algorithm builds a vector that includes all milestone information for a node. It aggregates these milestones as concentric circles. The circles have increasing order radii and different "intensities" corresponding to the number of milestones that are at a certain distance from the node. The Node Histogram algorithm generates an anonymous profile that does not carry the specific node's identity.

### C. Bayesian Techniques

In our framework, we use Bayesian Networks as the underlying learning and inferencing technique. The block diagram of our proposed estimation Bayesian algorithm is depicted in Figure 3. In describing the Bayesian algorithm, with a slight abuse of notation, we are going to refer to the Bayesian network nodes as components in order to avoid confusion with the use of the word node to denote participating machines on the physical network. Thus, expanding the Bayesian network, as shown in Figure 3, Block 3 has the profiles of the nodes as input, and is a continuous Gaussian component. In addition, Block 2 is a hidden binary component, and Block 1 is the output component acting as a $T$-class classifier. Thus, the output of the Bayesian network is a $T$-dimensional vector representing the probability distribution of the $T$ different classes. In the case of hop numbers estimation $T = 32$ corresponding to the hop numbers between the two input nodes. Note that this Bayesian network structure is a simple classical structure, often used in other applications of Bayesian Networks, where we have one component for each of the input and output in addition to a hidden node. The goal of the hidden component is to capture the latent relationships. We also experiment with a more complex structure as presented in Figure 4.

For example, if we need to estimate the distance from node $i$ to node $j$, we use the measurements from node $i$ to the landmarks and those from the landmarks to node $j$ as the input to the profiling module. This first module will create the respective profiles of $i$ and $j$ to feed into the Bayesian estimation algorithm, and the second module of our system will output a decision vector. The decision vector is basically the probability distribution of the distance being in each of all the possible distance values. Thus, in our system, the estimated distance is actually the position (or index) of the maximum value in the $T$-dimensional output vector. Following the example of the previous section of having 12-hops as our maximum, the input to the Bayesian module consists of the profile of node $i$ using measurements to the landmarks, while the profile of node $j$ is based on measurments from the landmarks. The Bayesian network then processes these two profiles using the learned behavior from the training data, and as a result, outputs a 12-dimensional vector representing the probability distribution of the distance from $i$ to $j$. An example of such a vector is $P_{i,j} = [0, 0.156, 0, 0.135, 0.15, 0.05, 0.254, 0.1, 0.1, 0.005, 0, 0.05]$. In this case, the maximum of the vector is 0.254 and is the seventh element of the vector, thus, the estimate of the distance, as reported by the Bayesian network, from node $i$ to node $j$ is 7 hops.

As we mentioned before, the structure of the Bayesian Network, as presented in Figure 3, is one of the most simple standard ones. As the nodes in the system and the result of their measurements became more complex, the Simple Bayesian Network failed to capture all the characteristics of the network. Thus, we extend the structure of the Bayesian network, as presented in Figure 4, where we divide the input into two vectors corresponding to the profiles of the two nodes in question. We add another hidden block for the newly introduced input node. This modification of the structure of the Bayesian network takes into consideration the fact that the input consists of two independent vectors, being the two profiles of the two nodes in question, and treats these two vectors as two separate variables. Note that following the notation of Bayesian Networks, Figures 3 and 4 assume that the processing flow is bottom-up, where the input feeds Block 3 and Block $3'$ and the output is generated by Block 1. We compare the performance of both Bayesian network structures in Section IV. In our implementation, we used the Bayes Net Toolbox (BNT) [1] on Matlab 7.0.1 [10].

## IV. Evaluation

We built a prototype of Minerva and evaluated its performance based on measurement data collected using Planet-lab. The current prototype can infer two network path metrics:

(1) latency and (2) number of hops between any two Planetlab nodes. In this section, we discuss the network metrics measurements we conducted and then present the results of evaluating and tuning the parameters of Minerva.

### *A. Measurement Setup*

The evaluation was done using measurements collected on the PlanetLab [14] platform that involved all 580 machines participating in the network as of August 2005. We deployed a modified version of the scriptroute suite of tools [16], where we removed the restrictions on the number of simultaneous measurements that exist in the default distribution. Our measurements engine, on every node, runs once every 8 hours collecting information using 3 tools, namely ping, traceroute and rockettrace (a modified version of traceroute that ships with scriptroute), targeted towards all other nodes on PlanetLab. While conducting these measurements, each engine on every node, independently chooses a random starting point from the list of the PlanetLab nodes; this was essential so that our massive measurements will not be mistaken for a DDOS (Distributed Denial of Service) attack and ensure that nodes are not in sync when sending their probing packets to any specific node which would interfer with the measurement results.

In the initial set of experiments, we choose a subset of our PlanetLab measurements consisting of 113 nodes and 11 landmarks distributed as follows: 2 in Europe, 2 in Asia, 1 in South America, 4 on the East coast, 1 on the West coast and 1 in the Middle of the US. We also use the simple Bayesian network structure presented in Figure 3. We use a small fraction of the actual collected measurements for training the Bayesian network, corresponding to 500 sample random measurements, which adds up to 3.95% of all possible measurements of $N(N-1)$ for $N = 113$. This will ensure that Minerva, as a system, does not introduce a large overhead of measurements. Note that the 113 nodes were chosen to represent a high percentage of all PlanetLab sites, since a typical site has several nodes. We then achieve our goal of straining the algorithm using a heterogeneous and diverse set of nodes, by starting with these 113 nodes and increasing the number to include all 580 PlanetLab nodes from our measurements.

When it comes to latency, defining the histogram of nodes requires us to take a closer look at the data as the measurements are not discrete values, as is the case of the hop numbers. Based on the obtained latencies, we used a granularity of 1 msec for latencies less than 50 msec, 10 msec for latencies between 50 msec and 500 msec, and a granularity of 50 msec for latencies greater than 500 msec. We also grouped all latencies greater than 1200 msec together as one bin. This results in a vector whose dimension is 111 points. Note that deciding on each group and its granularity is tunable and can be modified based on the application requirements.

### *B. Evaluation Metrics*

Besides the absolute error in inference, we use the accuracy of the proximity estimation of a given metric to evaluate the performance of our system. The accuracy metric captures how well the system can rank nodes in terms of their proximity (either for the number of hops or the latency) to a specific node. Assuming that an algorithm returns a set of $k$ nodes as the closest estimates (we use the term "closest" when dealing with latency or hop number proximity) for a certain node $i$ that we denote by $S_k^i$. Let the actual closest node to node $i$ be node $j$. Thus, the accuracy is 1 if $j \in S_k^i$ and 0 otherwise. The $k$-accuracy of an algorithm is computed as the presence of the closest node $j$ to a certain node $i$ in the set of the $k$ closest nodes as returned by the estimation system. More formally, it is defined as follows:

$$a(i) = \begin{cases} 1, & j \in S_k^i \\ 0, & otherwise \end{cases} \tag{2}$$

Note that in many practical situations, a node $i$ will query the inference system for the $k$ closest nodes. Then, node $i$ will perform its own measurements to this set of nodes. The reasoning behind this is that the estimation mechanism is basically providing the $k$ possible candidates of closest nodes and it is up to the node $i$ to perform its own measurements to determine the actual closest among this set of nodes. Thus, it is essential for the inference system to provide the querying node $i$ with its actual closest node among the returned $k$ nodes while maintaining $k << N$. Note that if $k$ is comparable in magnitude to $N$ then the whole purpose of an estimation system is defeated since the node $i$ is launching $k$ additional measurement on the network and the system cannot scale. Note that as $k$ increases, by definition, the accuracy increases. The goal is to achieve as high an accuracy with as low $k$ as possible.

In this section, we compare the accuracy obtained by Minerva, using the profiling and estimation modules, as presented in Section III, to the Vivaldi and Min-Sum algorithms, for different subsets of the collected data in terms of metrics of interest, number of nodes and landmarks. We also study the sensitivity of the system to different parameters such as training set, measurement overhead, and size of network.

### *C. Minerva Inference Performance*

*1) Accuracy:* Comparing the accuracy for latency estimation of Minerava, using the Node Histogram profiling algorithm and the Modified Bayesian Estimation module, to Min-Sum and Vivaldi, we observe the results in Figure 5. In the measurements we obtained from PlanetLab, some of the nodes were not responsive most of the time, if not always. For Vivaldi and Min-Sum, we disregard these unreliable nodes and omit them from the analysis, assuming the existence of a filtering mechanism. However, we do not offer the same filtering for the Bayesian Network estimator, expecting it to recognize such nodes on its own with its inherent probabilistic properties. The results shown in Figure 5 demonstrate clearly that the Bayesian Network estimator is able to estimate distance among nodes

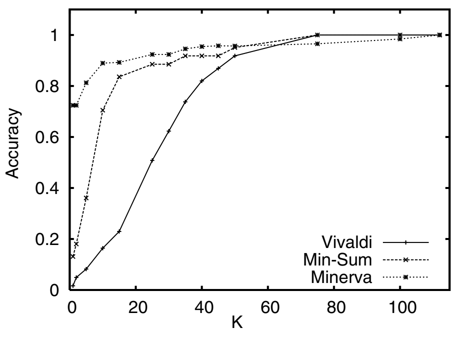

Fig. 5. Comparison of the Algorithms for Latency Estimation

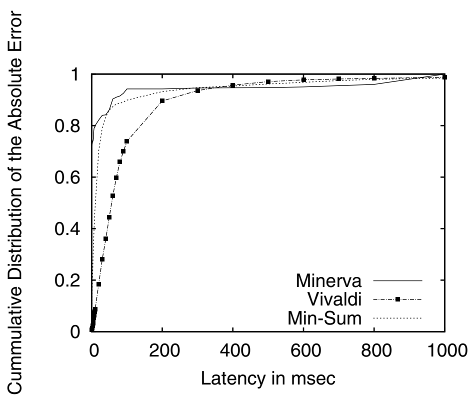

Fig. 7. Latency Absolute Error

and pick closest nodes much more precisely than Vivaldi and Min-Sum. In fact, for a small value of $k = 1$, the Bayesian Network system provides an accuracy of over 70%, while Vivaldi is at 1.6% and Min-Sum at 13%; a clear advantage of the Bayesian Network system. In addition, for $k = 10$, the Bayesian Network accuracy is at 88.9% compared to 16.4% for Vivaldi and 70.5% for Min-Sum. One point, though, worth noting, is that as $k$ goes over 50, this advantage seems to switch, and the Bayesian Network system seems to behave the worst among the three algorithms. This is due to the advantage we gave Min-Sum and Vivaldi by performing the filtering described earlier. However, we argue that, for most practical applications, choosing a high value of $k$, such as 50 or more, is not desirable, since the list returned to node $i$ of possible candidates is too long to be considered a useful answer.

Repeating the same experiment for hop-count estimation, Figure 6 shows the results for the accuracy of Minerva's hop-count estimation and the Min-Sum algorithm. As was the case with latency estimation, Minerva is highly accurate in estimating the hop-counts between two nodes.

*2) Absolute Error:* We plot the absolute error of the latency estimations in Figure 7. Note that Minerva does not provide a numerical value for latency instead it is a pointer to a bin. Our bins are explained at the beginning of this section. If the estimator points to the same bin that holds the actual value, we assume that the error is 0, otherwise the error is calculated as the absolute value of actual value – center of the bin. For example, if Minerva estimates the latency to belong to bin 3 corresponding to $[2 - 3]$ msec, while the actual value is 5.1 msec, then the absolute error is computed as 5.1-2.5 = 2.6 msec. While the accuracy, as a metric, shows that the answers that Minerva provides can be used to rank node distances with high probability correctly, Figure 7 shows that the estimates are often representative of the actual latency values among nodes since around 80% of the estimates had an absolute error of less than 10 msec.

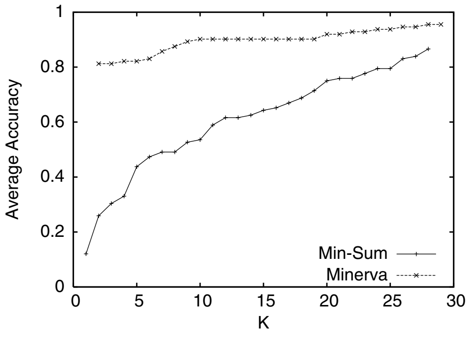

Fig. 6. Comparison of the Algorithms for Hop Count Estimation

### *D. Scalability and Parameter Sensitivity*

This section presents experiments targeted at understanding the sensitivity of Minerva to several parameters of the system, and its scalability to changes in the participating nodes. We also include results aimed at characterizing the effects of using a machine learning algorithm, more specifically Bayesian networks, in our inference engine.

*1) Number of Landmarks:* An essential parameter to study is the scaling effect of landmarks as the number of nodes increases. In order to quantify the effect of the number of landmarks, we present in Figure 8 the latency estimation accuracy as we add nodes to the initial set of 113. The figure shows that there is no noticeable decrease in the accuracy of the estimates up till the number of nodes reaches 300. At that point, an increase in the number of landmarks results in higher levels of accuracy. Note though, that an increase of over 209% in the number of nodes, only requires a modest increase of around 36% in the number of landmarks. As such, we can claim that indeed the number of landmarks does not need to increase in the same proportion as the number of nodes does.

We observe similar behavior for the hop-count estimation, as can be seen in Figure 12. Other inference algorithms such as Min-Sum and GNP experience similar effects on accuracy as the number of landmarks is increased.

*2) Bayesian Network Classifier:* As described in Section III-C, we explored two Bayesian network classifiers for Minerva. Figure 9 plots the hop-count accuracy using the simple Bayesian Network classifier presented in Figure 3 and compares it to the Modified Bayesian Network classifier of Figure 4. We can see how the Modified Bayesian Network is able to characterize the nodes with a higher accuracy. The reason behind this lies in the fact that the two input histograms represent two different nodes, and treating them as separate

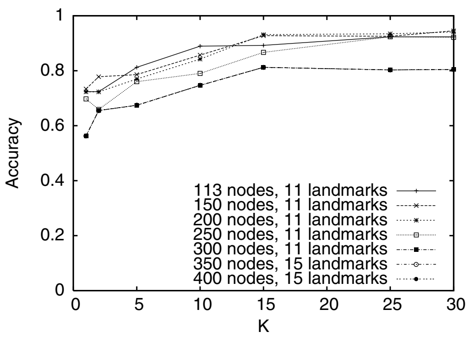

Fig. 8. The Effect of Landmarks on Estimating Latency

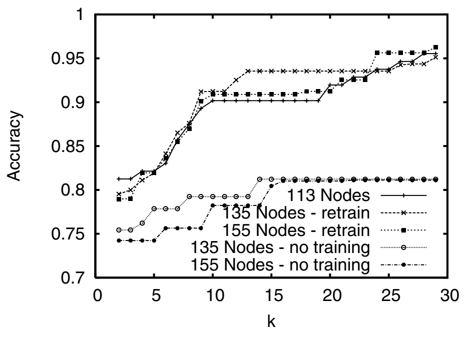

Fig. 10. Effect of Initial Training Set and Number of Nodes on Hop-Count Accuracy

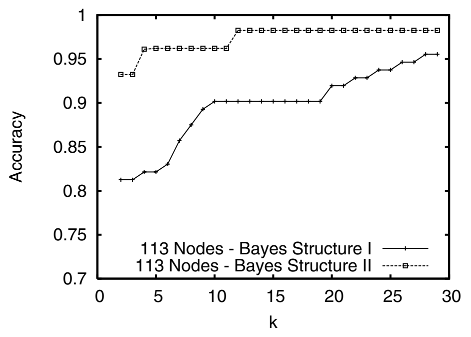

Fig. 9. Effect of Bayesian Network Structure on Accuracy

input variables makes it easier for the Bayesian network classifier to characterize them, and extract their probabilistic properties.

*3) Retraining and Addition of New Nodes:* In what follows, we study the effect of increasing the network size on the accuracy. We consider two scenarios: at first, we increase the number of nodes and measure the accuracy of the system, then we re-train the system in order to include the newly added nodes and compare the results. This set of experiments aims at understanding the cost of using a machine learning algorithm and, more specifically, the frequency of its training phase. Figure 10 depicts the accuracy of the tested networks for both scenarios of re-train and no re-train for hop count. We observe that re-training indeed does improve the accuracy. However, as we will see next, this is mainly due to the fact that the network that we used for the initial training (113 nodes) was too small to yield enough information that can fully characterize the rest of the nodes in the system. As the initial network size that is used for training increases, we note that we can continue to use the obtained Bayesian Network classifier for larger networks, without re-training, since the data was enough to capture the specifics of the topology of the network as a whole.

Note that as nodes are added to the network, new milestones might emerge. These can be either routers that never appeared before or routers that had appeared only once before the new addition of nodes, thus did not qualify prior to this addition to become milestones. In this case, we update the histograms of the affected nodes to reflect the new milestones, despite the fact that we may have used the old histograms of these nodes for the training of the Bayesian Network classifier. In fact, we argue in here that this change does not affect the classifier since the signature-like profiles of our system do not contain the identity of the respective nodes and is meant to capture a snapshot of the network characteristics.

In this set of experiments, we start with a subset of the network consisting of 200 nodes and 15 landmarks. We extract the signatures of the nodes and use 2000 samples for training the Bayesian Network classifier. Note that we increase the number of samples used for training as we increase the number of nodes, however, the percentage is still modest compared to the full $N^2$ measurements of 40000. Figure 11 shows the accuracy of the classifier versus $k$ for hop count. Then we increase the number of nodes in the network and re-measure the accuracy of the classifier without re-training the classifier. We also show the accuracy for the nodes that were added in each experiment to the initial network of 200 nodes. By measuring the accuracy for these nodes, we are, actually, testing how well the Bayesian Network classifier is able to generalize rules from the initial observed data (i.e. that of the initial 200 nodes) and use these observations to infer the behavior of the newly introduced nodes.

As in any learning-based system, we need to train the Bayesian Network classifier. This training is quite costly in terms of computation resources, and requires end-to-end measurements to be used for training. Thus, for the system to be scalable, we need to keep this training to a minimum versus the dynamics of the network as a whole, such as the addition of nodes to the system. For example, we want a system that does not need to be re-trained every time a node joins or leaves. We now increase the number of nodes in our set and re-train for every set of experiments. We plot the results of the accuracy in Figure 12 showing that the accuracy does not deteriorate as we increase $N$ and with a slight increase in the number of landmarks $L$ the percentage of correct classification depicted in the accuracy remains in the same range showing that the algorithm is able to characterize the topology correctly.

Figure 13 shows the effect of increasing the number of nodes while using the results from the initial set of nodes and the effect of re-training. While retraining with data and measurements obtained for the added nodes improves the accuracy, we note that the system did tolerate the introduction of

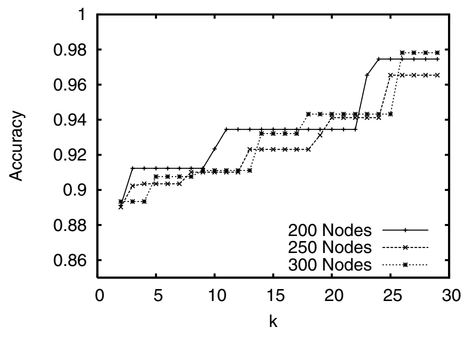

Fig. 11. Hop-Count Accuracy (System Trained for 200 Nodes)

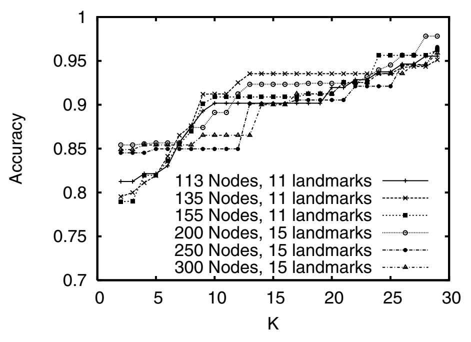

Fig. 12. Hop-Count Accuracy vs. Number of Nodes in the system

these nodes without a significant deterioration in its accuracy, especially for a small value of $k$. These results show that Minerva is able to extend the previously observed behavior of node to newly introduced nodes joining the network.

*4) Bayesian Approximation Iterations:* By definition, the Bayesian Network algorithm relies on likelihood maximization leading to the use of iterative approximation techniques [6], [8]. Since the time required for training and, often, the quality of the estimator are directly proportional to the number of iterations during training, we wanted to study and understand its effect on our system. In this section, we test the performance of the whole system of profiling and estimation as we change the number of iterations allowed during the training stage of the Bayesian Network estimator. Figure 14 shows the

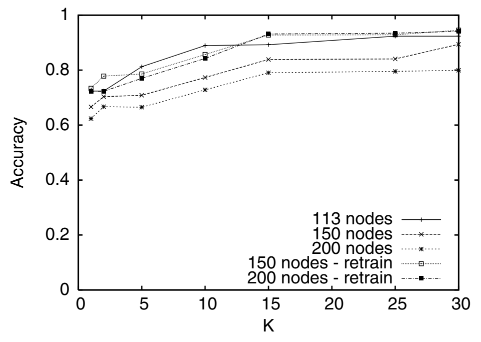

Fig. 13. The Effect of Re-Training on Estimating Latency

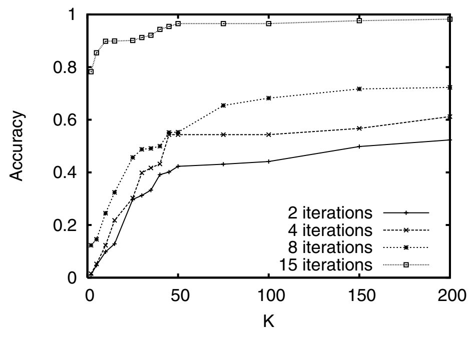

Fig. 14. Hop-Count Accuracy vs. Number of Iterations During Training

accuracy plotted for the different values of $k$ as we vary the number of iterations. The network used for this experiment consists of 552 nodes and 22 landmarks. We evaluate the accuracy for 2, 4, 8 and 15 iterations during the training stage for hop count estimation. We observe that for this larger set of nodes, a small number of iterations does not provide a high accuracy for a small value of $k$. In fact, the accuracy for $k = 2$ was below 15% for the 2, 4 and 8 iterations. However, as we increase the number of iterations to 15, the accuracy jumps to around 80%, a major improvement. What happens in here is due to the fact that Bayesian Network maximum likelihood is trying to maximize its function and, just like any other learning mechanism, uses these iterations to refine its parameters. This behavior is not an artifact of our proposed system, but is a normal behavior of any system that relies on Bayesian Networks.

## *E. Inference of Time Varying Metrics*

Using the collected data, we compare latency among the same pairs of nodes over time. As expected, the variations are quite considerable. Figure 15 shows the standard deviation of differences in latency for the same set of pair of nodes. The figure tells us that over 60% of measurements of pairs of nodes in the network had a standard deviation greater than 100 msec, which can be considered as extreme variations with time. Most of the current inference mechanisms do not take time variation into consideration, as part of the system design. This basically means that nodes have to continue to repeat their measurements over the time in order to determine its closest neighbors at any instance. Minerva, on the other hand, can leverage the learning capability to accurately learn and infer time variation of different metrics. To validate this capability, we now show how Minerva can be tailored and trained to infer inter-node latencies over time. We expand the profiling vectors of nodes to include two flags: the first indicating whether the specific measurement was taken on a week day or a weekend day, the second indicating the time period when the measurement was taken as morning, afternoon, or night. This translates into an expanded profile vector of 113 values corresponding to the 111 vector, presented above, and the 2 new flags. Since the first flag is mainly binary and the second one can take one of three possible values,

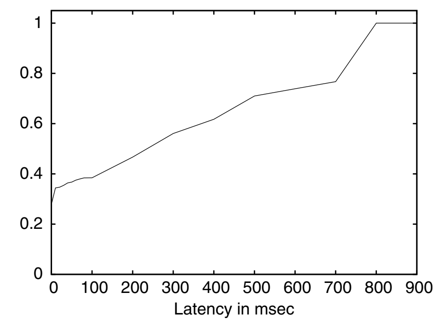

Fig. 15. Cummulative Distribution of the St-Dev of Latencies Over Time Showing the Large Variations

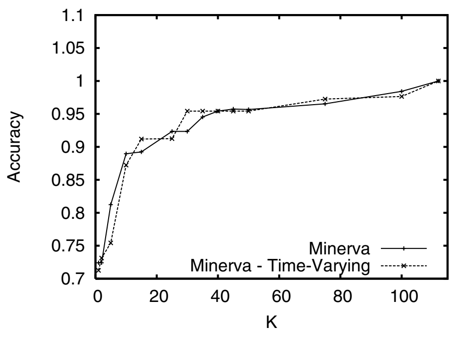

Fig. 16. Inferring Latencies Over Time

we end up with a total of six combinations. We repeat the training of the Bayesian Network estimation module using 3000 samples. We start with the same set of 500 samples used in the experiment where time variations were not considered and use six measurements corresponding to the six different combinations. We then test the estimation for the whole network by computing the accuracy of querying Minerva using profiles that contain the 2 time flags. The results in Figure 16 show that our Bayesian Network estimator with the use of the Node Histogram algorithm can estimate latencies and infer their changes with time, with a high accuracy, a feature that other latency and distance estimators do not consider. In fact, the two plots in the figure are comparable showing that expanding the profiles to include time information allowed Minerva to learn about the behavior and continue to produce reliable estimates. Note that the flags can be different and can include further details of the latency changes such as hourly, if the need be.

## V. Future Work & Conclusions

The importance of accurately and efficiently estimating locality of services and computing network distances between different nodes has significantly increased due to proliferation of p2p networks and is also evident from the abundance of latency estimation schemes. Similar to applications' use of network latency to improve the download performance, the number of hops between nodes can be potentially used as a measure for path reliability. In this paper, we have presented a learning based estimation approach for network and node metrics that relies on probabilistic techniques, more specifically Bayesian Networks. To the extent of our knowledge, our use of anonymous profiles coupled with learning techniques is quite unique. We showed encouraging results leading us to conclude that this approach is quite promising and can be used in different applications.

## References

[1] Bayes Net Toolbox (BNT). http://bnt.sourceforge.net/.

[2] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, J. Chase, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," Operating Systms Design and Implementation (OSDI), San Francisco, December 2004.

[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. "Vivaldi: A Decentralized Network Coordinate System," In the Proceedings of the ACM SIGCOMM '04 Conference, Portland, Oregon, August 2004.

[4] R. Fonseca, P. Sharma, S. Banerjee, S.J. Lee, S. Basu, "Distributed Querying of Internet Distance Information," IEEE Global Internet Symposium (in conjunction with InfoCom 2005), Miami, Florida March 2005.

[5] P. Francis, S. Jamin, C. Jin, D. Raz, Y. Shavitt, L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," IEEE/ACM Trans. on Networking, Oct. 2001.

[6] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian Network Classifiers," Machine Learning 29:131–163, 1997.

[7] K. P. Gummadi, S. Saroiu, S. D. Gribble, "King: Estimating latency between arbitrary Internet end hosts," Proceedings of SIGCOMM IMW 2002, November 2002, Marseille, France.

[8] G. R. Iversen, *Bayesian Statistical Inference.* Sage University Papers Series, Quantitative Applications in the Social Sciences ; No. 07-043. Beverly Hills, Calif. Sage Publications, Inc., 1984.

[9] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, A. Venkataramani, "iPlane: An Information Plane for Distributed Services," OSDI 2006, November 2006.

[10] Matlab. http://www.mathworks.com/products/ matlab/.

[11] T. S. E. Ng, H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches", Proceedings of IEEE INFOCOM'02, New York, June 2002.

[12] V. Padmanabhan, L. Qiu, and H. Wang, "Server-based Inference of Internet Link Lossiness," In Proceedings of IEEE INFOCOM'03, San Francisco, CA, USA, April 2003.

[13] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, S. Bhatti, "Lighthouses for Scalable Distributed Location," IPTPS '03.

[14] PlanetLab. http://www.planet-lab.org/.

[15] S. Ratnasamy, M. Handley, R. Karp, S. Shenker, "Topologically-Aware Overlay Construction and Server Selection," Proceedings of Infocom 2002.

[16] Scriptroute. http://www.cs.washington.edu/research/networking/scriptroute/.

[17] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee, "Estimating Network Proximity and Latency," ACM SIGCOMM Computer Communications Review, Vol. 36, No. 3, pp 41-50, July 2006.

[18] H. H. Song, L. Qiu, and Yin Zhang, "NetQuest: A Flexible Framework for Large-Scale Network Measurement," in Proceedings of the ACM SIGMETRICS Conference, Saint-Malo, France, June 2006.

[19] S. Srinivasan and E. Zegura, "M-coop:A Scalable Infrastructure for Network Measurement," Third IEEE Workshop on Internet Applications (WIAPP '03).

[20] S. Srinivasan, and E. Zegura, "Network Measurement as a Cooperative Enterprise," IPTPS '02.

[21] L. Tang, and M. Crovella, "Virtual Landmarks for the Internet," Internet Measurement Conference Oct 2003.

[22] B. Wong, A. Slivkins, and E.G. Sirer, "Meridian: A Lightweight Network Location Service Without Virtual Corrdinates," In the Proceedings of the ACM SIGCOMM '05 Conference, Philadelphia, Pennsylvania, August 2005.

[23] P. Yalagandula, P. Sharma, S. Banerjee, Sung-Ju Lee, and S. Basu, "S$^3$: A Scalable Sensing Service for Monitoring Large Networked Systems," in Proceedings of the SIGCOMM Workshop on Internet Network Management, Pisa, Italy, August 2006.